11. Rock, Paper, Scissors Write a program to play a three-game match of "rock, paper, scissors" between a person and a computer. See Fig. 7.15. The program should use a class named *Contestant* having two subclasses named *Human* and *Computer*. After the person makes his or her choice, the computer should make its choice at random. The *Contestant* class should have instance variables for *name* and *score*. (***Note:*** *Rock* beats *scissors*, *scissors* beats *paper*, and *paper* beats *rock*.)

11.

```
import random

def main():
    ## Play three games of rock, paper, scissors.
    # Get names of contestants and instantiate an object for each.
    nameOfHuman = input("Enter name of human: ")
    h = Human(nameOfHuman)
    nameOfComputer = input("Enter name of computer: ")
    c = Computer(nameOfComputer)
    print()
    # Play three games and keep score.
    for i in range(3):
        humanChoice = h.makeChoice()
        computerChoice = c.makeChoice()
        print("{0} chooses {1}".format(c.getName(), computerChoice))
        if humanChoice == "rock":
            if computerChoice == "scissors":
                h.incrementScore()
            elif computerChoice == "paper":
                c.incrementScore()
        elif humanChoice == "paper":
            if computerChoice == "rock":
                h.incrementScore()
            elif computerChoice == "scissors":
                c.incrementScore()
        else:  # humanChoice = scissors
            if computerChoice == "rock":
                c.incrementScore()
            elif computerChoice == "paper":
                h.incrementScore()
        print(h, end="  ")
        print(c)
        print()
    if h.getScore() > c.getScore():
        print(h.getName().upper(), "WINS")
    elif c.getScore() > h.getScore():
        print(c.getName().upper(), "WINS")
    else:
        print("TIE")
```

```
class Contestant():
    def   init  (self, name="", score=0):
        self._name = name
        self._score = score

    def getName(self):
        return self._name

    def getScore(self):
        return self._score

    def incrementScore(self):
        self._score += 1

    def   str  (self):
        return "{0}: {1}".format(self._name, self._score)

class Human(Contestant):
    def makeChoice(self):
        choices = ["rock", "paper", "scissors"]
        while True:
            choice = input(self._name + ", enter your choice: ")
            if choice.lower() in choices:
                break
        return choice.lower()

class Computer(Contestant):
    def makeChoice(self):
        choices = ["rock", "paper", "scissors"]
        selection = random.choice(choices)
        return selection

main()
```

```
Enter name of human: Garry
Enter name of computer: Big Blue

Garry, enter your choice: rock
Big Blue chooses scissors
Garry: 1  Big Blue: 0

Garry, enter your choice: scissors
Big Blue chooses paper
Garry: 2  Big Blue: 0

Garry, enter your choice: rock
Big Blue chooses rock
Garry: 2  Big Blue: 0

GARRY WINS
```

12. Semester Grades Redo Example 2 so that each subclass has its own __str__ method, and therefore the program will illustrate polymorphism with the __str__ methods. Assume that all letter-grade students are full-time students, but that pass–fail students might be either full-time or part-time. The output of the program should give each student's name, grade, and status. See Fig. 7.16.

```
Enter student's name: Bob
Enter student's grade on midterm exam: 79
Enter student's grade on final exam: 85
Enter category (LG or PF): LG
Do you want to continue (Y/N)? Y
Enter student's name: Alice
Enter student's grade on midterm exam: 92
Enter student's grade on final exam: 96
Enter category (LG or PF): PF
Are you a full-time student (Y/N)? N
Do you want to continue (Y/N)? N

NAME    GRADE   STATUS
Alice   Pass    Part-time student
Bob     B       Full-time student
```

FIGURE 7.16 Possible outcome of Exercise 12.

```
12. def main():
        ## Calculate semester grades.
        listOfStudents = obtainListOfStudents()   # students and grades
        displayResults(listOfStudents)

    def obtainListOfStudents():
        listOfStudents = []
        carryOn = 'Y'
        while carryOn == 'Y':
            name = input("Enter student's name: ")
            midterm = float(input("Enter student's grade on midterm exam: "))
            final = float(input("Enter student's grade on final exam: "))
            category = input("Enter category (LG or PF): ")
            if category.upper() == "LG":
                st = LGstudent(name, midterm, final)
            else:
                status = input("Are you a full-time student (Y/N)? ")
                if status.upper() == 'Y':
                    fullTime = True
                else:
                    fullTime = False
                st = PFstudent(name, midterm, final, fullTime)
            listOfStudents.append(st)
            carryOn = input("Do you want to continue (Y/N)? ")
            carryOn = carryOn.upper()
        return listOfStudents

    def displayResults(listOfStudents):
        print("\nNAME\tGRADE\tSTATUS")
        listOfStudents.sort(key = lambda x: x.getName())
        for pupil in listOfStudents:
            print(pupil)
```

```
class Student:
    def   init  (self, name="", midterm=0, final=0):
        self._name = name
        self._midterm = midterm
        self._final = final
        self._semesterGrade = ""

    def setName(self, name):
        self._name = name

    def setMidterm(self, midterm):
        self._midterm = midterm

    def setFinal(self, final):
        self._final = final

    def getName(self):
        return self._name

    def   str  (self):
        return self._name + "\t" + self.calcSemGrade()
class LGstudent(Student):

    def calcSemGrade(self):
        average = round((self._midterm + self._final) / 2)


        if average >= 90:
            return "A"
        elif average >= 80:
            return "B"
        elif average >= 70:
            return "C"
        elif average >= 60:
            return "D"
        else:
            return "F"

    def   str  (self):
        return (self._name + "\t" + self.calcSemGrade() +
                "\tFull-time student")
```

1. United Nations The file **UN.txt** gives data about the 193 members of the United Nations. Each line of the file contains four pieces of data about a country—*name*, *continent*, *population* (in millions), and *land area* (in square miles). Some lines of the file are

```
Canada,North America,34.8,3855000
France,Europe,66.3,211209
New Zealand,Australia/Oceania,4.4,103738
Nigeria,Africa,177.2,356669
```

```
Pakistan,Asia,196.2,310403
Peru,South America,30.1,496226
```

(a) Create a class named *Nation* with four instance variables to hold the data for a country and a method named *popDensity* that calculates the population density of the country. Write a program that uses the class to create a dictionary of 193 items, where each item of the dictionary has the form

name of a country: Nation object for that country

Use the file **UN.txt** to create the dictionary, and save the dictionary in a pickled binary file named **nationsDict.dat**. Also, save the class *Nation* in a file named **nation.py**.

(b) Write a program that requests the name of a U.N. member country as input, and then displays information about the country as shown in Fig. 7.20. Use the pickled binary file **nationsDict.dat** and the file **nation.py** created in part (a).

```
Enter a country: Canada
Continent: North America
Population: 34,800,000
Area: 3,855,000.00 square miles
```

FIGURE 7.20 Possible outcome of Prog. Project 1(b).

```
Enter a continent: South America
  Ecuador
  Colombia
  Venezuela
  Brazil
  Peru
```

FIGURE 7.21 Outcome of Prog. Project 1(c).

(c) Write a program that requests the name of a continent as input, and then displays the names (in descending order) of the five most densely populated U.N. member countries in that continent. See Fig. 7.21. Use the pickled binary file **nationsDict.dat** and the file **nation.py** created in part (a).

1(a).
```
import pickle

def main():
    createDictionayOfNations()

def createDictionayOfNations():
    nationDict = {}
    for line in open("UN.txt", 'r'):
        data = line.split(',')
        country = Nation()
        country.setName(data[0])
        country.setContinent(data[1])
        country.setPopulation(float(data[2]))
        country.setArea(float(data[3].rstrip()))
        nationDict[country.getName()] = country
    # Save list as binary file.
    pickle.dump(nationDict, open("nationDict.dat", 'wb'))
    return nationDict

class Nation:
    def   init  (self):
        self_name = ""
        self_continent = ""
        self_population = 0.0
        self_area = 0

    def setName(self, name):
        self._name = name

    def getName(self):
        return self._name

    def setContinent(self, continent):
        self._continent = continent

    def getContinent(self):
        return self._continent

    def setPopulation(self, population):
        self._population = population

    def getPopulation(self):
        return self._population

    def setArea(self, area):
        self._area = area

    def getArea(self):
        return self._area

    def popDensity(self):
        return self._population / self._area

main()
```

1(b).
```
import pickle
from nation import Nation

def main():
    ## Display information about a country.
    nationDict = pickle.load(open("nationDict.dat", "rb"))
    country = input("Enter a country: ")
    print("Continent:", nationDict[country].getContinent())
    print("Population: {0:,.0f}".
          format(1000000 * nationDict[country].getPopulation()))
    print("Area: {0:,.2f} square miles".
          format(nationDict[country].getArea()))

main()
```

```
Enter a country: Canada
Continent: North America
Population: 34,800,000
Area: 3,855,000.00 square miles
```

1(c).
```
import pickle
from nation import Nation

def main():
    ## Display the most density populated countries on a continent.
    nationDict = pickle.load(open("nationDict.dat", "rb"))
    nationList = list(nationDict.keys())
    continent = input("Enter a continent: ")
    nationsInContinent = [nation for nation in nationList if
              nationDict[nation].getContinent() == continent]
    nationsInContinent.sort(key=lambda x: nationDict[x].popDensity(),
                            reverse=True)
    for i in range(5):
        print(nationsInContinent[i])

main()
```

```
Enter a continent: South America
  Ecuador
  Colombia
  Venezuela
  Peru
  Brazil
```

2. Savings Account Write a program to maintain a savings account. The program should use a class named *SavingsAccount* with instance variables for the customer's name and the account balance, and two methods named *makeDeposit* and *makeWithdrawal*. The *makeWithdrawal* method should deny withdrawals that exceed the balance in the account. See Fig. 7.22.

```
2. def main():
       acct = SavingsAccount()
       name = input("Enter person's name: ")
       acct.setName(name)
       print("D = Deposit, W = Withdrawal, Q = Quit")
       request = input("Enter D, W, or Q: ").upper()
       while True:
           if request == 'D':
               amount = float(input("Enter amount to deposit: "))
               acct.makeDeposit(amount)
               print("Balance: ${0:,.2f}".format(acct.getBalance()))
               request = input("Enter D, W, or Q: ").upper()
           elif request == 'W':
               amount = float(input("Enter amount to withdraw: "))
               acct.makeWithdrawal(amount)
               print("Balance: ${0:,.2f}".format(acct.getBalance()))
               request = input("Enter D, W, or Q: ").upper()
           elif request == 'Q':
               print("End of transactions. Have a good day",
                     acct.getName() + '.')
               break
           else:
               request = input("Enter D, W, or Q: ").upper()

   class SavingsAccount:
       def   init  (self, name="", balance=0.0):
           self._name = name
           self._balance = balance
       def setName(self, name):
           self._name = name
       def getName(self):
           return self._name
       def setBalance(self, balance):
           self._balance = balance
       def getBalance(self):
           return self._balance
       def makeDeposit(self, amount):
           self._balance += amount
       def makeWithdrawal(self, amount):
           if amount <= self._balance:
               self._balance -= amount
           else:
               print("Insufficient funds, transaction denied.")
```

```
Enter person's name: Fred
D = Deposit, W = Withdrawal, Q = Quit
Enter D, W, or Q: D
Enter amount to deposit: 1000
Balance: $1,000.00
Enter D, W, or Q: W
Enter amount to withdraw: 4000
Insufficient funds, transaction denied.
Balance: $1,000.00
Enter D, W, or Q: W
Enter amount to withdraw: 400
Balance: $600.00
Enter D, W, or Q: Q
End of transactions. Have a good day Fred.
```

3. Cab Fare Management Write a program for a cab owner to display the complete list of rides for his cabs. He has both sedans and hatchbacks. The program should contain a class named Cab having two subclasses named Sedan and Hatchback. The Cab class should have instance variables for type of cab and number of kilometers driven. Each subclass should have a caculateFare method. The fare for sedans should be $2 per kilometer and the fare for hatchbacks should be $1.5 per kilometer.

After the data for all cabs has been entered, the program should display the total kilometers driven for both types of cabs. The program should also display the total kilometers driven for all cabs, and the total fare earned from all cabs. See Fig 7.23. The program should use a list of objects.

```
Enter cab type(Hatchback/Sedan): hatchback
Enter the number of kilometers travelled: 10
Do you want to continue (Y/N)? Y
Enter cab type(Hatchback/Sedan): Sedan
Enter the number of kilometers travelled: 12
Do you want to continue (Y/N)? N

-----Kilometers driven for each cab-----
Hatchback: 10 kilometers
Sedan: 12 kilometers
-----------------------

Total number of kilometers driven by all Cabs: 22.0
Total fare earned from all cabs (in dollars): 39.0
```

FIGURE 7.23 Possible outcome of Programming Project 3.

```
3. def main():
       listOfDrives = createListOfDrives()
       displayResults(listOfDrives)

   class Cab:
           def __init__(self, cabType="", numberOfKilometers=0.0):
               self._cabType = cabType
               self._numberOfKilometers = numberOfKilometers
           def setName(self, cabType):
               self._cabType = cabType
           def getName(self):
               return self._cabType
           def setNumberOfKilometers(self, numberOfKilometers):
               self._numberOfKilometers = numberOfKilometers
           def getNumberOfKilometers(self):
               return self._numberOfKilometers

   class Sedan(Cab):
           def calculate(self):
               rate = 2.0
               return float(self.getNumberOfKilometers()) * rate

   class Hatchback(Cab):
           def calculate(self):
               rate = 1.5
               return float(self.getNumberOfKilometers()) * rate
```

```
def createListOfDrives():
        listOfDrives = []
        carryOn = 'Y'
        while carryOn == 'Y':
            name = input("Enter cab type(Hatchback/Sedan):
").capitalize()
            if name!='Sedan' and name!='Hatchback':
                print("Please enter either Sedan or Hatchback!")
                continue

            kilometerTravelled = input("Enter the number of kilometers
travelled: ")
            if name == "Sedan":
                cab = Sedan(name, kilometerTravelled)
            elif name == "Hatchback":
                cab = Hatchback(name, kilometerTravelled)


            listOfDrives.append(cab)
            carryOn = input("Do you want to continue (Y/N)? ")
            carryOn = carryOn.upper()
        return listOfDrives

def displayResults(listOfDrives):
        print()
        totalFare = 0
        totalNumbersOfKilometers = 0.0
        print("-----List of Drives-----")
        for drive in listOfDrives:
            print(str(drive.getName())+ " : " +
str(drive.getNumberOfKilometers()) + " kilometers")
        print("------------------------")
        for drive in listOfDrives:
            totalNumbersOfKilometers = totalNumbersOfKilometers +

        int(drive.getNumberOfKilometers())
                    if (drive.getName() == "Sedan"):
                        totalFare += drive.calculate()
                    elif (drive.getName() == "Hatchback"):
                        totalFare += drive.calculate()
                print("Total number of kilometers drived by all Cabs: " +
        str(totalNumbersOfKilometers))
                print("Total Fare earned from all cabs (in dollars): " +
        str(totalFare))
```